NACHNAME:	**SEMESTER:** ☐ **M5** ☐ **M6** ☐ M3 ☐ M4 ☐ M7
VORNAME:	**VERTIEFUNG:** ☐ **FV** ☐ **IM**

VERWENDETE KLASSEN:	• Als **Anlage** erhalten Sie einen Ausdruck des vorab bekannt gemachten Quelltextes von verschiedenen Klassen, insbesondere von mehreren Versionen einer Klasse `Wort`.

UNBEDINGT BEACHTEN:	• **Bevor** Sie mit der Bearbeitung beginnen, müssen die Angaben zur Person auf dieser Seite **vollständig** ausgefüllt sein. • Es sind **keinerlei Hilfsmittel** zugelassen.

GENERELLE VORGABEN:	• Es sind keinerlei Kommentare verlangt, weder `javadoc`-Kommentare noch andere. • Methoden müssen vollständig, auch mit Annotationen, angegeben werden. Die umgebende Klasse und etwaige `import`–Anweisungen werden nicht verlangt. • Die einzelnen Zeichen – auch Groß- oder Klein-Schreibung – müssen klar erkennbar sein. • Programmier-Richtlinien (insbesondere Checkstyle) sind zu beachten. Bei Testklassen dürfen aber *magic numbers* verwendet werden.

Aufgabe 1: (32 Punkte)

a) Ergänzen Sie jeweils in den eckigen Klammern:

Bei einer [ ] Klasse ist es oftmals nötig, statt dem Original eine Kopie weiterzureichen. Eine diesbezügliche Empfehlung von Joshua Bloch besagt: „Machen Sie bei Bedarf [ ]".

Klassen mit dem Qualitäts-Standard [ ] ermöglichen dieses Kopieren durch Überschreiben der Methode [ ].

Bei `version2` von `Wort` kann ein `Wort w` kopiert werden durch

```
Wort kopie = [ ];
```

Ohne [ ] Rückgabetyp wäre ein expliziter Cast nötig:

```
Wort kopie = [ ];
```

b) Die Methode `entsprichtMuster` der Klasse `MusterStrings` prüft, ob ein String einem bestimmten Muster entspricht. Darauf aufbauend sollen zwei zusätzliche Methoden der Klasse entstehen:

Die Methode `entsprichtEinem(String s, String... muster)` soll prüfen, ob der angegebene String `s` (mindestens) einem der angegebenen Muster genügt.

Die Methode `waehleEntsprechende(List<String> l, String... muster)` soll in einer Liste `l` von Strings alle Einträge löschen, die nicht (mindestens) einem der angegebenen Muster genügen.

Schreiben Sie eine (einzige) JUnit-Test-Methode, welche zunächst für jeden der zehn Strings `"eins", "zwei", "drei", "vier", "fuenf", "sechs", "sieben", "acht", "neun", "zehn"` das Ergebnis von `entsprichtEinem(s, "????", "?????")` überprüft.

In der gleichen Test-Methode soll für eine aus den obigen zehn Strings gebildete Liste `l` getestet werden, dass `waehleEntsprechende(l, "e???")` in der Liste nur `"eins"` übrig lässt und `waehleEntsprechende(l, "?e??")` die beiden Strings `"neun"` und `"zehn"`.

c) Implementieren Sie die beiden Methode `entsprichtEinem` und `waehleEntsprechende`:

Aufgabe 2: (20 Punkte)

a) Im **Konstruktor** der Version 3 von `Wort` sollte (wie in Version 1) durch Werfen von Exceptions verhindert werden, dass ein illegales Wort entsteht. Nehmen Sie die entsprechende Verbesserung im Konstruktor der **Version 3** von `Wort` vor:

```
public Wort(String... w) {
```

```
}
```

b) Auch in den Methoden `setString` und `rueckgaengig` der **Version 3** von `Wort` können Ausnahme-Situationen entstehen.

Schreiben Sie eine Test-Methode, welche für diese beiden Methoden (nicht für den Konstruktor) bei **je einem** unerlaubten Aufruf das Werfen der **richtigen**, also die Fehlerursache benennenden `Exception` überprüft.

```
@Test
public void testeExceptions() {
```

```
}
```

Aufgabe 3: (30 Punkte)

Die Aufgabe bezieht sich (unter anderem) auf die **drei Versionen** der Klasse `Wort` und die Methode `tuWasMit` der Klasse `BleedsGschwaetz`.

Alle von `Wort` abhängigen Klassen beziehen sich auf dieselbe Version. Zur gleichen Zeit wird demnach immer nur eine Version der Klasse `Wort` verwendet.

a) An geeigneter Stelle werde die Klasse `Name` wie folgt definiert:

```
package aufgabe2;
import anhang.version?.Wort; // ? durch jeweilige Version ersetzen
import anhang.*;

public class Name extends Wort {
    private Anrede mAnrede;
    public Name(String s, Anrede a) {
        super(s);
        mAnrede = a;
    }
    @Override
    public boolean equals(Object vergl) {
        if (!super.equals(vergl)) return false;
        if (!(vergl instanceof Name)) return false;
        Name n = (Name) vergl;
        return n.mAnrede == mAnrede;
    }
}
```

Irgendwo werden drei Variablen initialisiert:

```
Name n1a = new Name("A", FRAU);
Name n1b = new Name("A", FRAU);
Name n2 = new Name("A", HERR);
```

Geben Sie an, welchen Wert dann die in der linken Spalte angegebenen Ausdrücke damit jeweils haben. Abkürzungen sind nicht erlaubt. Tragen Sie **Exc.** ein, wenn der Aufruf zu einer `Exception` führt.

	version1	version2	version3
`tuWasMit(n1a, n1b, n2)`			
`tuWasMit(n1a, n1a, n2)`			
`tuWasMit(n1a, n2, null)`			
`tuWasMit(n2, n2, null)`			

b) Im dieser Teilaufgabe geht es um die Klasse `BenutzerName`:

```
package aufgabe2;
import anhang.*;

public class BenutzerName extends Name {
    private boolean mGast;
    protected BenutzerName(String s, Anrede a, boolean gast) {
        super(s, a);
        mGast = gast;
    }
    public BenutzerName(String s, Anrede a) {
        this(s, a, false);
    }
    public boolean equals(Object vergl) {
        if (!super.equals(vergl)) return false;
        BenutzerName b = (BenutzerName) vergl;
        return b.mGast == mGast;
    }
}
```

In nachstehender Tabelle seien:

```
BenutzerName b1 = new BenutzerName("B", HERR, true);
BenutzerName b2 = new BenutzerName("B", FRAU, true);
BenutzerName b3 = new BenutzerName("B", HERR, false);
```

Tragen Sie wieder die Ergebnisse der angegebenen `tuWasMit`-Aufrufe in die Tabelle ein. Das Auftreten von `Exception`s soll wie zuvor mit **Exc.** markiert werden.

	version1	version2	version3
`tuWasMit(b1, b1, b2, null)`			
`tuWasMit(b1, b1, b3)`			
`tuWasMit(null, null, b3)`			

c) Schließlich wird die Klasse `GastBenutzer` betrachtet:

```
package aufgabe2;
import anhang.*;

public class GastBenutzer extends BenutzerName {
    public GastBenutzer(String s, Anrede a) {
        super(s, a, true);
    }
}
```

Irgendwo werden vier `Worte` initialisiert:

```
Wort w = new Wort("C");
Wort n = new Name("C", FRAU);
Wort b = new BenutzerName("C", FRAU);
Wort g = new GastBenutzer("C", FRAU);
```

Tragen Sie in nachstehender Tabelle (außer in den schwarzen Diagonalfeldern) jeweils das Ergebnis von `tuWasMit(x, y)` bei der **Version 2** von `Wort` ein. Das Auftreten von `Exception`s soll wie zuvor mit **Exc.** markiert werden.

Version 2	y = w	y = n	y = b	y = g
x = w				
x = n				
x = b				
x = g				

Aufgabe 4: (8 Punkte)

Die Aufgabe bezieht sich auf die Klasse `BleedsGschwaetz` der Anlage. Es wird die **Version 2** der Klasse `Wort` verwendet.

Die folgende Testmethode läuft merkwürdigerweise fehlerfrei ab:

```
@Test
public void testeEhrenWort() {
```

```
    Wort a1 = new Wort("unschuldig");
    Wort a2 = ehrenWort("unschuldig");
    assertTrue(a2.equals(a1));
    a2.setString("schuldig"); // a)
    assertTrue(a2.equals(a1));
    a1.setString("schuldig"); // b)
    assertFalse(a2.equals(a1));
}
```

a) Erläutern Sie **kurz** (in Stichworten), warum `a2` nach der Veränderung (erste markierte Zeile) immer noch gleich `a1` ist.

b) Erläutern Sie **kurz**, warum dies nach der entsprechenden Veränderung von `a1` (zweite markierte Zeile) dann nicht mehr gilt.